Model-Free Methods in Reinforcement Learning: Learning Directly from Experience

Ahmed BADI
ahmedbadi905@gmail.com
linkedin.com/in/badi-ahmed

January 2026

Abstract

Model-free methods in reinforcement learning (RL) allow an agent to learn good behavior directly from experience, without ever building an explicit model of the environment's dynamics. Instead of estimating transition probabilities or reward functions, the agent updates value functions or policies based only on observed states, actions, and rewards. This article provides a clear and intuitive introduction to model-free RL. We start with value-based methods such as Monte Carlo evaluation, SARSA, and Q-learning, and then move to policy gradient methods including the REINFORCE algorithm, actor–critic, and the Asynchronous Advantage Actor–Critic (A3C) algorithm. For each method, we explain the main intuition, present the core update equations, and discuss advantages and limitations compared to model-based approaches.

Keywords: Model-Free Reinforcement Learning, Q-learning, SARSA, Monte Carlo Methods, REINFORCE, Actor–Critic, A3C.

1 Introduction

In reinforcement learning, an agent interacts with an environment, takes actions, and receives rewards. Over time, it aims to learn a strategy (policy) that maximizes long-term reward. Many algorithms in RL are *model-free*: they do not learn or use an explicit model of the environment's transition dynamics. [1], [2]

1.1 What Does Model-Free Mean?

In an MDP, the model consists of:

- Transition probabilities $P(s' \mid s, a)$.
- Reward function $R(s, a, s')$.

Model-free methods:

- Do not estimate P or R explicitly.
- Learn value functions and/or policies directly from sampled transitions (s, a, r, s').

[1], [3]

1.2 Why Model-Free Methods?

Model-free RL is attractive because:

- It avoids the complexity of learning a full model.
- It is often simpler to implement and widely applicable.

However, model-free methods usually require more *real* interactions with the environment than model-based methods, so they can be less sample-efficient. [2], [4]

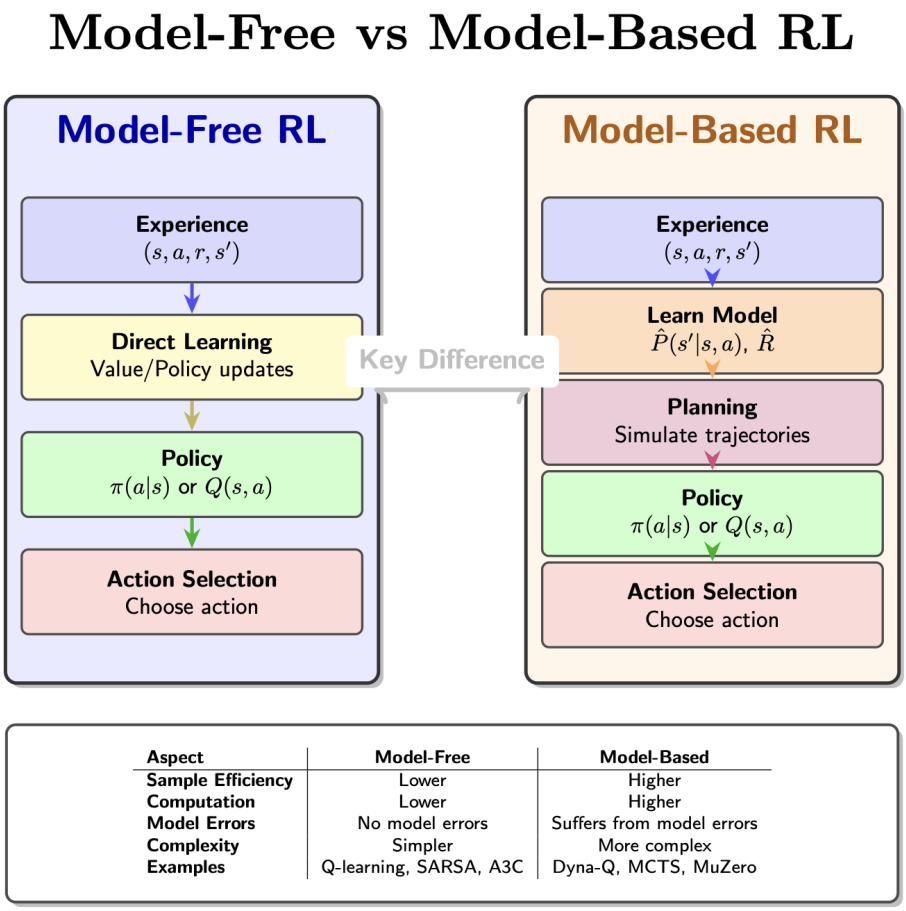


Figure 1: High-level comparison: model-free RL learns directly from experience; model-based RL uses a learned or known model to plan.

2 Background: MDPs, Policies, and Returns

We briefly recall the MDP setting and value functions to fix notation. [3], [5]

2.1 MDP Setting

An MDP is defined by $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, P transition probabilities, R rewards, and $\gamma \in [0, 1)$ the discount factor.

A policy $\pi(a \mid s)$ gives the probability of choosing action a in state s. The return from time t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}.$$

2.2 Value Functions

For a policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s], \qquad q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a].$$

The aim is to find an optimal policy $\pi^\star$ maximizing expected return, often via optimal value functions $v_\star$ or $q_\star$. [3]

3 Monte Carlo Methods

Monte Carlo (MC) methods estimate value functions from complete episodes of experience without requiring a model of transitions. [3], [6]

3.1 Monte Carlo State-Value Estimation

For a policy π, MC estimates $v_\pi(s)$ as the average of returns following visits to s in sampled episodes:

$$v_\pi(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)}(s),$$

where $G^{(i)}(s)$ is the return after the i-th visit to state s, and $N(s)$ is the number of visits. [3]

Incremental update form:

$$v_\pi(s) \leftarrow v_\pi(s) + \alpha \left(G - v_\pi(s)\right),$$

where G is the observed return and α is a step size.

3.2 Monte Carlo Control

MC can also be used for control (finding $\pi^\star$):

- Maintain action-value estimates $Q(s,a)$.
- Generate episodes using an exploring policy (e.g., ϵ-greedy).
- For each state–action pair, average the returns following its occurrences.
- Improve the policy by making it greedy with respect to Q.

This is a model-free way to approximate $q_\star(s,a)$ using complete episodes. [3], [6]

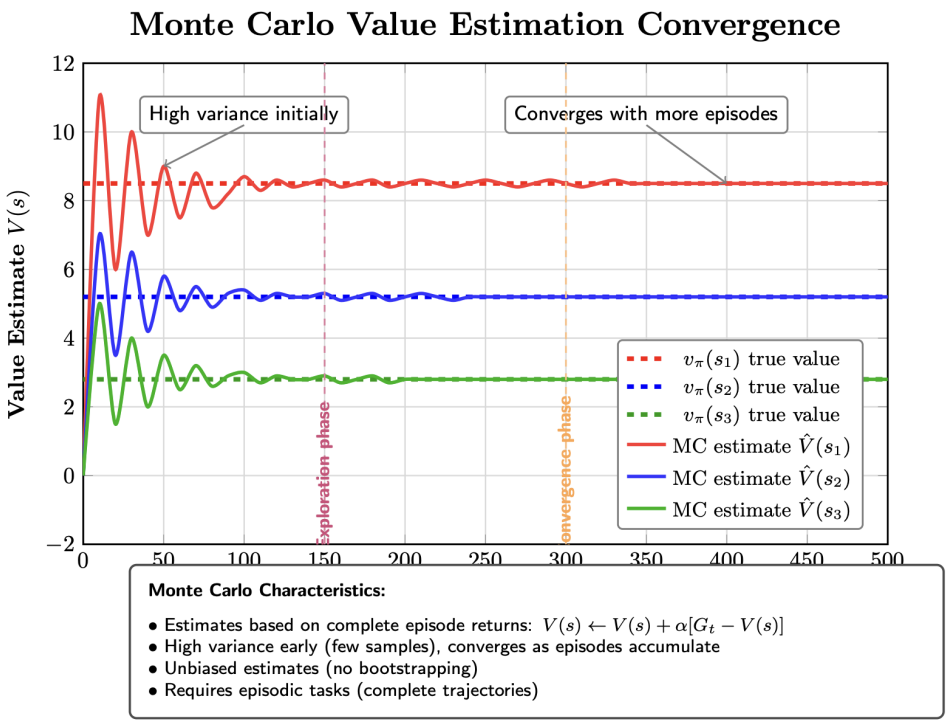


Figure 2: Illustrative convergence of Monte Carlo value estimates over episodes for a small MDP.

4 Temporal-Difference Control: SARSA and Q-Learning

Temporal-Difference (TD) methods update value estimates at each step using bootstrapping (next-state estimates) instead of waiting for episode end. [3], [7]

4.1 SARSA: On-Policy TD Control

SARSA is an on-policy TD control algorithm that learns $Q(s, a)$ while following the same policy it evaluates. [3], [8]

Given a transition $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, the update is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right].$$

Key properties:

- **On-policy**: action A_{t+1} is drawn from the current (often ϵ-greedy) policy.
- Learns the value of the policy that is actually used to generate behavior.

4.2 Q-Learning: Off-Policy TD Control

Q-learning is an off-policy TD control method that directly approximates the optimal action-value function $Q_\star(s, a)$. [3], [9]

Given transition $(S_t, A_t, R_{t+1}, S_{t+1})$, the update is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\right].$$

Key properties:

- **Off-policy**: the update uses the greedy action $\arg\max_{a'} Q(S_{t+1}, a')$ even if behavior is ϵ-greedy.
- Converges to $Q_\star$ under suitable conditions (learning rate, exploration). [3]

4.3 Exploration with ϵ-Greedy Policies

Both SARSA and Q-learning typically use ϵ-greedy exploration:

$$A_t = \begin{cases} \text{random action}, & \text{with prob. } \epsilon, \\ \arg\max_a Q(S_t, a), & \text{with prob. } 1 - \epsilon. \end{cases}$$

This balances:

- **Exploration**: try new actions to discover their values.
- **Exploitation**: choose the best-known action to maximize reward.

[3]

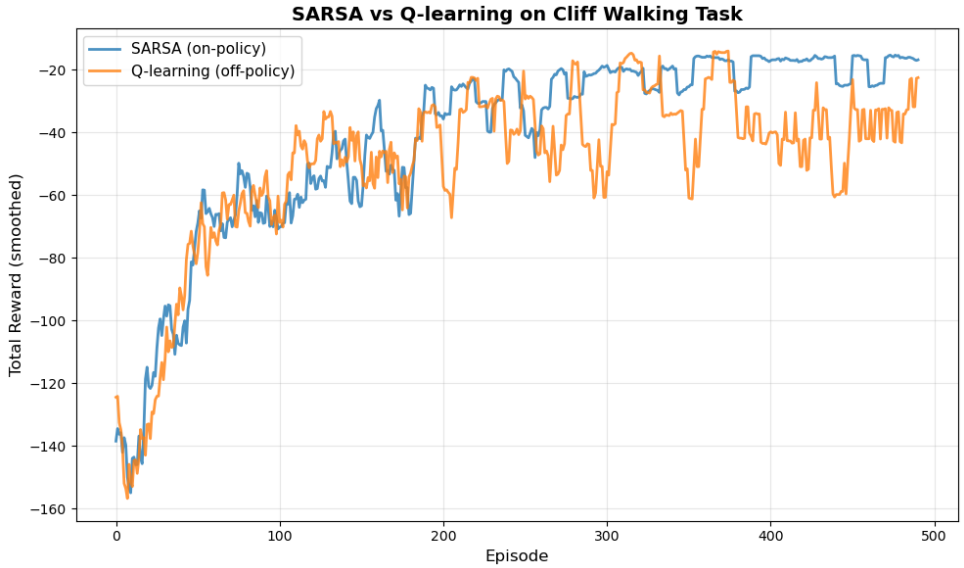
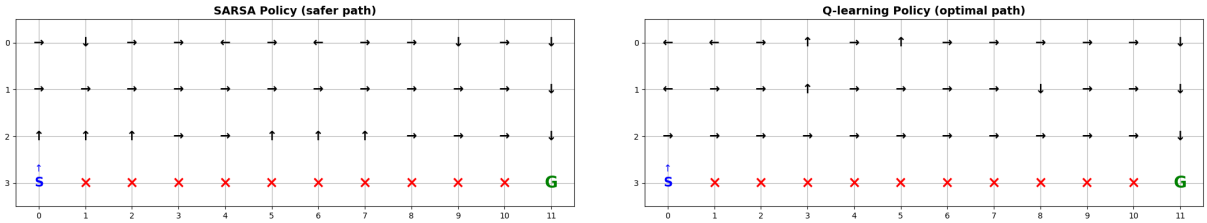


Figure 3: Typical comparison of SARSA and Q-learning on the cliff-walking task. SARSA tends to learn safer policies, while Q-learning converges to the optimal but riskier path.



(a) SARSA policy (safer path)

(b) Q-learning policy (optimal path)

Figure 4: Comparison of learned policies. SARSA (left) learns a safer route avoiding the cliff edge, while Q-learning (right) learns the optimal but riskier path along the cliff. S=Start, G=Goal, ×=Cliff. Arrows indicate the best action in each state.

5 Summary of Core Model-Free Value-Based Methods

Table 1 summarizes the main characteristics of Monte Carlo, SARSA, and Q-learning.

Method	**What it estimates**	**Needs episodes?**	**On- or off-policy?**
Monte Carlo	$v_\pi(s)$ or $q_\pi(s,a)$ from returns	Yes (complete episodes)	On-policy (typically)
SARSA	$Q_\pi(s,a)$ of the behavior policy	No (step-by-step)	On-policy
Q-Learning	$Q_\star(s,a)$ (optimal)	No (step-by-step)	Off-policy

Table 1: Key model-free value-based RL methods.

6 Policy Gradient Methods

Value-based methods learn value functions and derive policies from them. Policy gradient methods instead directly parameterize the policy and optimize its parameters by gradient ascent on expected return. [3], [10]

6.1 Parameterized Policy

Let $\pi_\theta(a \mid s)$ be a policy with parameters θ. The objective is to maximize

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_0].$$

Policy gradient methods estimate $\nabla_\theta J(\theta)$ and update

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta).$$

7 The REINFORCE Algorithm

REINFORCE is a classic Monte Carlo policy gradient algorithm. It updates policy parameters at the end of episodes using sampled returns. [3], [11]

7.1 Basic Update Rule

For an episodic task, the gradient estimator is

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t \mid S_t)\, G_t,$$

and the parameter update is

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t \mid S_t)\, G_t.$$

Intuition:

- Actions that lead to higher return get their probabilities increased.
- Actions that lead to lower return get their probabilities decreased.

7.2 Variance Reduction with Baselines

REINFORCE can have high variance. A common variance reduction trick is to subtract a baseline $b(s)$ (often a value function):

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(A_t \mid S_t)\,(G_t - b(S_t)).$$

This does not change the expected gradient but can reduce variance and speed up learning. [3], [10]

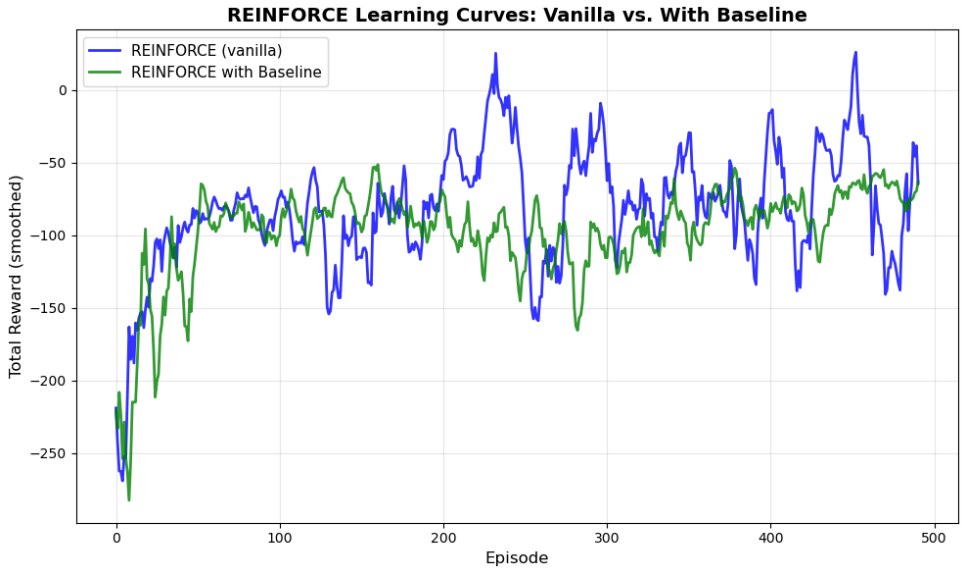


Figure 5: Learning curves comparing vanilla REINFORCE with REINFORCE using a baseline. The baseline version shows faster convergence and more stable learning.

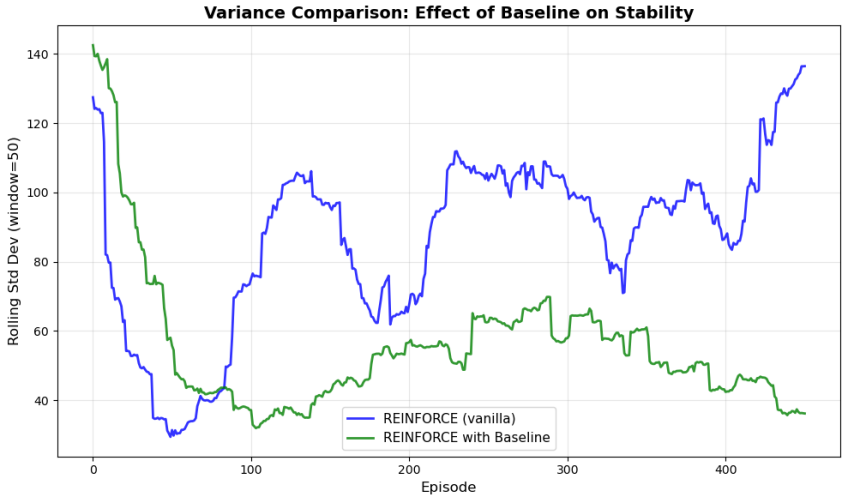


Figure 6: Variance comparison between vanilla REINFORCE and REINFORCE with baseline. The baseline significantly reduces variance, demonstrating the effectiveness of this variance reduction technique.

8 Actor–Critic Methods

Actor–critic methods combine policy gradients (actor) with value function approximation (critic). The critic reduces variance by providing a learned baseline or advantage estimate. [3], [12]

8.1 Actor and Critic Roles

- **Actor**: parameterized policy $\pi_\theta(a \mid s)$ that selects actions.
- **Critic**: value function $V_w(s)$ (or $Q_w(s, a)$) with parameters w that estimates expected return.

8.2 Advantage Actor–Critic Update

A common form uses the estimated advantage

$$A_t = R_{t+1} + \gamma V_w(S_{t+1}) - V_w(S_t).$$

Updates:

- Critic (TD update):

$$w \leftarrow w + \beta \left[R_{t+1} + \gamma V_w(S_{t+1}) - V_w(S_t)\right] \nabla_w V_w(S_t).$$

- Actor:

$$\theta \leftarrow \theta + \alpha \, A_t \, \nabla_\theta \log \pi_\theta(A_t \mid S_t).$$

This uses TD learning for the critic and a policy gradient update scaled by the advantage. [10], [12]

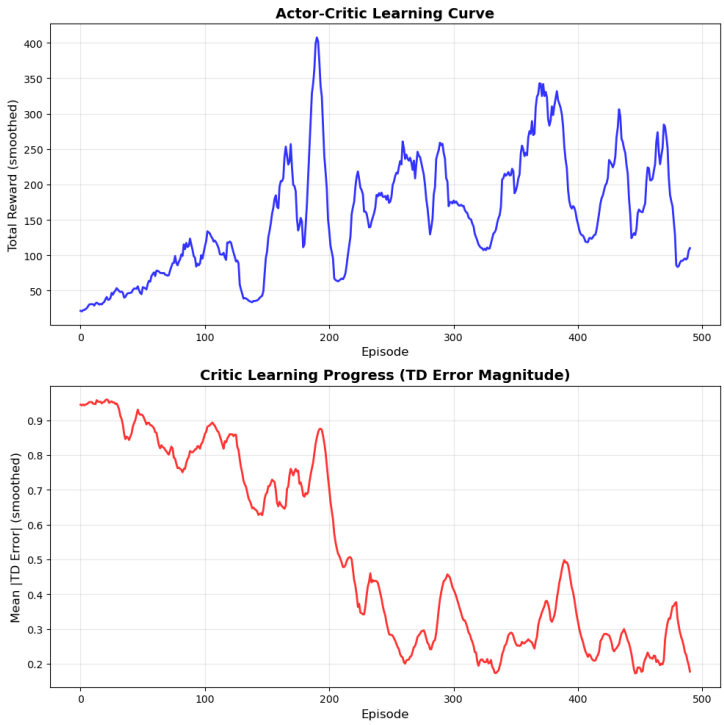


Figure 7: Actor-critic learning progress. Top: episodic rewards increase as the agent learns. Bottom: TD error magnitude decreases as the critic's value estimates improve, demonstrating convergence.

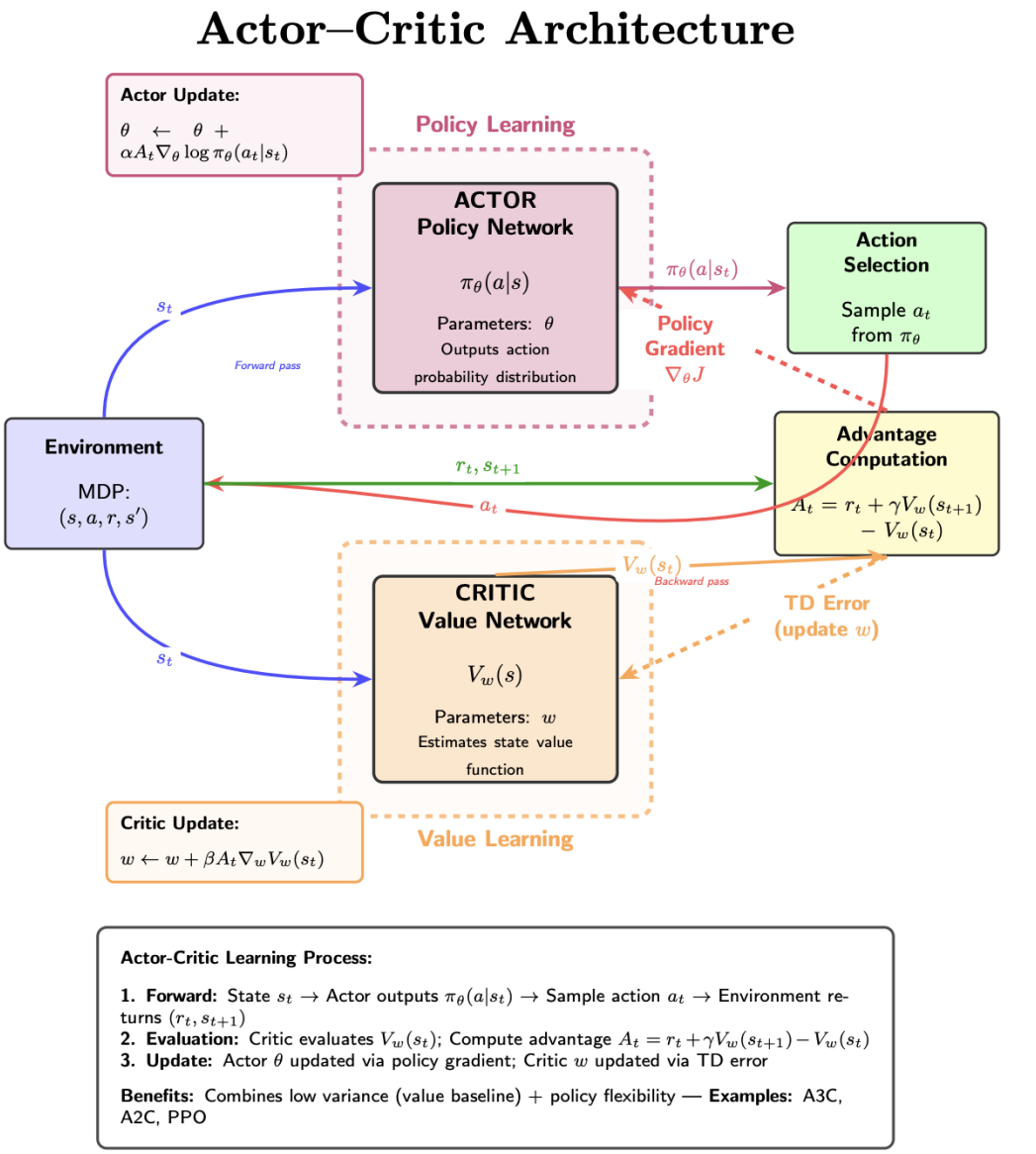


Figure 8: Actor–critic architecture: the actor outputs an action distribution, while the critic evaluates states and provides a learning signal.

9 Asynchronous Advantage Actor–Critic (A3C)

A3C is a deep RL algorithm that runs multiple actor–critic agents in parallel environments, updating a shared set of parameters asynchronously. It improves stability and training speed. [13], [14]

9.1 Key Ideas

- Multiple worker agents interact with independent copies of the environment.
- Each worker maintains local copies of actor and critic parameters.
- Workers periodically compute gradients and apply them to shared global parameters.
- Asynchronous updates help decorrelate data and reduce the need for experience replay.

9.2 A3C Loss Functions

Typical A3C combines three losses:

- **Policy loss** (actor) using advantage estimates.
- **Value loss** (critic) using squared TD error.
- **Entropy bonus** to encourage exploration.

[13], [14]

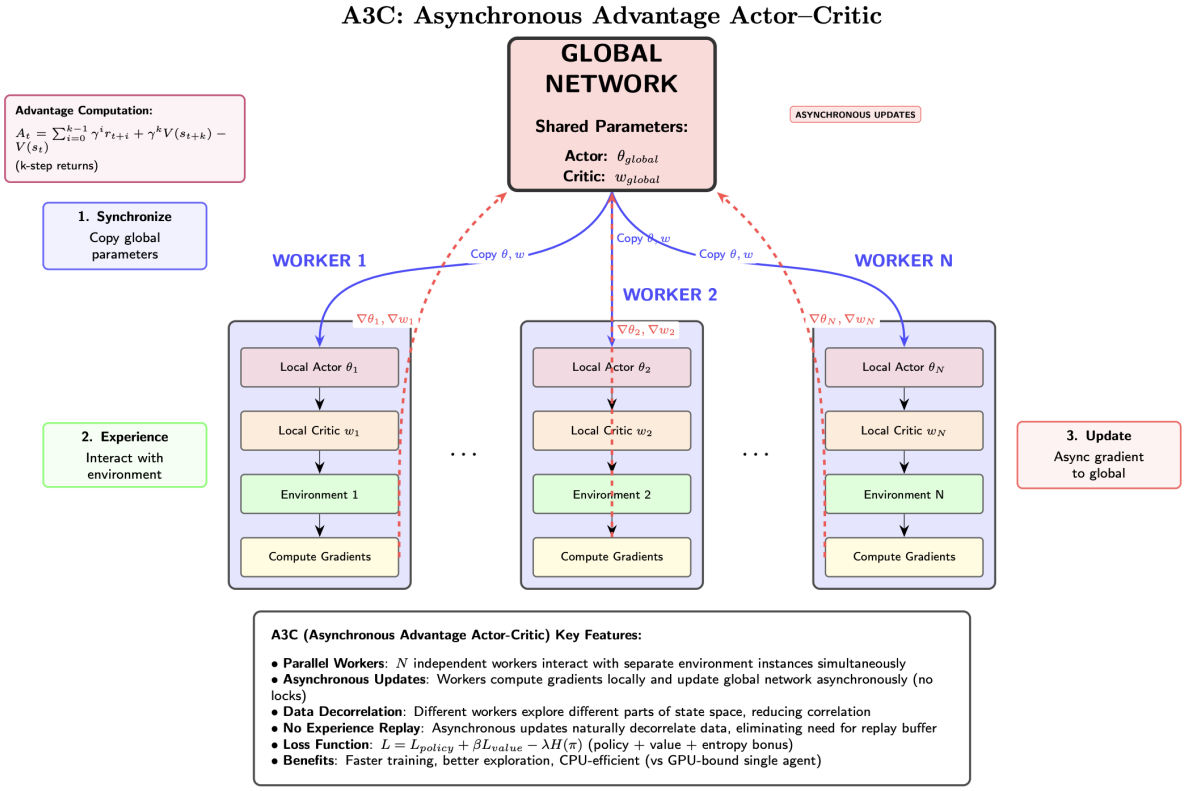


Figure 9: A3C architecture: multiple asynchronous actor–critic workers interact with their environments and update shared global parameters.

10 Model-Free vs Model-Based: Trade-Offs

Model-free and model-based RL offer different strengths. [2], [4]

10.1 Sample Efficiency vs Simplicity

- **Model-free**:
  - Simpler to implement; no need to learn a model.
  - Often needs many environment interactions (lower sample efficiency).
- **Model-based**:
  - More sample-efficient by reusing experience in simulated rollouts.
  - More complex and computationally expensive due to model learning and planning.

10.2 When to Choose Model-Free Methods

Model-free methods are often chosen when:

- Environment interactions are cheap (e.g., fast simulators).
- Dynamics are too complex to model accurately.
- Implementation simplicity is a priority.

[1], [15]

11 Conclusion

Model-free methods form the backbone of many reinforcement learning applications. This article has:

- Introduced Monte Carlo methods, SARSA, and Q-learning as core value-based algorithms.
- Explained the REINFORCE algorithm as a basic policy gradient method.
- Presented actor–critic and A3C as more advanced model-free approaches combining value estimation and policy optimization.

These tools provide a strong foundation for practical RL, especially when combined with function approximation and deep neural networks in modern deep reinforcement learning.

References

[1] GeeksforGeeks, *Model-free reinforcement learning: An overview*, 2024. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/model-free-reinforcement-learning-an-overview/

[2] GeeksforGeeks, *Differences between model-free and model-based reinforcement learning*, 2024. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/differences-between-model-free-and-model-based-reinforcement-learning/

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[4] Ericsson, *Reinforcement learning: Model-based vs. model-free*, 2023. [Online]. Available: https://www.ericsson.com/en/blog/2023/12/comparing-model-based-and-model-free-reinforcement-learning-characteristics-and-applications

[5] Wikipedia contributors, *Reinforcement learning*, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning

[6] B. Bouzy, *Apprentissage par renforcement (3): Monte carlo, td, q-learning and sarsa*, 2005. [Online]. Available: https://helios2.mi.parisdescartes.fr/~bouzy/Doc/AA2/ReinforcementLearning3.pdf

[7] D. Precup, *Mc control, sarsa, q-learning*, 2022. [Online]. Available: https://www.cs.mcgill.ca/~dprecup/courses/Winter2022/Lectures/7-TD-control-2022.pdf

[8] GeeksforGeeks, *Sarsa reinforcement learning*, 2020. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/sarsa-reinforcement-learning/

[9] GeeksforGeeks, *Q-learning in python*, 2019. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/q-learning-in-python/

[10] YouTube Tutorial, *Deep dive into reinforce, a2c, a3c & more (policy gradient methods)*, 2025. [Online]. Available: https://www.youtube.com/watch?v=007EvVofBC0

[11] GeeksforGeeks, *Reinforce algorithm*, 2020. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/reinforce-algorithm/

[12] GeeksforGeeks, *Actor-critic algorithm in reinforcement learning*, 2023. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/actor-critic-algorithm-in-reinforcement-learning/

[13] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," *International Conference on Machine Learning (ICML)*, 2016. [Online]. Available: https://arxiv.org/abs/1602.01783

[14] GeeksforGeeks, *Asynchronous advantage actor-critic (a3c) algorithm*, 2019. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/asynchronous-advantage-actor-critic-a3c-algorithm/

[15] Neptune.ai, *Model-based and model-free reinforcement learning: Pytennis case study*, 2024. [Online]. Available: https://neptune.ai/blog/model-based-and-model-free-reinforcement-learning-pytennis-case-study